

AUS920010023US1

Patent Application

Application for United States Patent

of

David H. Evans, *et al.*

for

5 Debugger Probe for Object Oriented Programming

CROSS-REFERENCE TO RELATED APPLICATIONS

(CLAIMING BENEFIT UNDER 35 U.S.C. 120)

Not applicable.

10 FEDERALLY SPONSORED RESEARCH

AND DEVELOPMENT STATEMENT

This invention was not developed in conjunction with any Federally sponsored contract.

MICROFICHE APPENDIX

15 Not applicable.

INCORPORATION BY REFERENCE

United States Patent Number 5,901,315, issued to Jonathan W. Edwards, David H. Evans, and Michael G Vassil, on May 4, 1999, is hereby incorporated by reference in its entirety, including description and diagrams.

20

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to computer software development tools and more particularly to a method and system for debugging a Java application
5 that includes native method dynamic load libraries (e.g., C or C++ code).

Description of the Related Art

Traditional programming methodologies are organized around actions and logic. Programs developed using traditional software design methodologies are usually easily represented by flow diagrams and could be viewed as a procedure of
10 steps which receive input, processes the data, and produces output.

Object Oriented Programming ("OOP"), however, is organized around objects and data. OOP approaches programming from a perspective centralized on providing objects, each object having an interface through which it may be used or by other programs and objects to perform a "method" or set of methods.

15 One of the earliest object-oriented computer languages was Smalltalk[TM]. More recent and perhaps more widely used object-oriented languages are C++ and Java.

The Java programming language is especially adapted for use in distributed applications, such as on corporate networks and the Internet. Additionally, Java
20 applications have found extensive use on the World Wide Web.

Java developers often "extend" a Java application by linking their own Java code with native method dynamic load libraries ("DLL"). A "native method" DLL is usually a function written in another programming language, such as "C" or "C++", which is not portable across different operating platforms without recompilation.

5 When the Java application includes linked native method DLL's, a problem arises in debugging the application. In particular, while "tools" for debugging Java code, on the one hand, and tools for debugging C/C++ code, on the other hand, are well-known, the art lacked the ability to simultaneously debug a piece of software that included both types of code.

10 As a result, some available methods addressed the problem of debugging Java applications having linked native method DLL's by using a "brute force" approach by using distinct debug routines that are run separately. This approach required the Java application with the linked native language DLL's to be executed twice, one for each debug routine, thereby substantially increasing the time necessary to debug the Java
15 application.

 The system and method disclosed in US Patent Number 5,901,315 to Edwards, *et al.*, provided a solution to this problem at the time of its availability and in keeping with currently available companion Java development tools. This system and method provided for debugging a target application which comprised Java code having linked
20 native method DLL's which was carried out in a computer having an operating system, a system debug application programming interface ("API"), and a Java virtual machine ("JVM") having a Java debug API.

According to the Edwards system and method, the JVM was first launched under the system debug API, and the Java application was then run under the JVM. Because the JVM that runs the target application itself runs under the system debug API, simultaneous control of the target application via the system debug API and the Java debug API was enabled. Thus, the method allowed the debug of the target application by simultaneously debugging the Java code and the native method dynamic load libraries. Events generated during the debug process were characterized as being initiated from the Java code or the native method DLL's, as the case may be. The preferred embodiment of the Edwards debugger comprised a graphical user interface ("GUI") front end, a debug engine back end, and a "probe" or daemon. The GUI provided an interface through which the user made requests and viewed the status of the application being debugged. The debug engine performed the debugging work by controlling the probe to manipulate the target application through the JVM, and to report events to the debug engine.

The probe was preferably implemented as two distinct processes, a first "daemon" process that performed native method debugging, and a second "probe" process that performed the Java method debugging. The first process also preferably controlled the second process across the system debug API and communicated therewith via a socket connection to facilitate the simultaneous debugging of the Java and C/C++ code comprising the target application.

According to another feature of the Edwards method, the front and back end components of the debugger were supported on a first computer, while the first and

second "processes" were supported on a second computer. The first and second computers could then be connected over a computer network such as a TCP/IP connection to facilitate remote debugging of the target application. The Edwards system and method was implemented in a programming tool product from

- 5 International Business Machines ("IBM") under the name Interactive Code Analysis Tool ("ICAT").

The current ICAT for Java utilizes a GUI to permit source-level debugging of Java applications on Microsoft's Windows[TM] NT 4.0 and Windows2000. Using ICAT, application developers can set breakpoints, execute and step their applications, and examine the application's stack and variables all with the click of a mouse. If desired, the debugger can be operated remotely; that is, ICAT resides on one system, and the probe resides on the system containing the application. Communication between the two systems is via a TCP/IP connection. The ICAT probe is written in Java, and it interacts with the JVM as needed to control the application being

10 debugged. Essentially, ICAT starts a Windows process, which is the JVM running the probe. ICAT then communicates with the probe via TCP/IP. If ICAT is being run remotely, a daemon is run on the target system which interacts with the probe.

For versions of the JVM before version 1.3, the ICAT probe utilized Sun Microsystems' sun.tools.debug Application Programming Interface ("API") in order

20 to control the application being debugged. The sun.tools.debug API is easy to use, but it is missing a number of functions that the ICAT customarily provided for other debuggers for other programming languages.

Beginning with version 1.3 of the JVM, several new debugger interfaces were provided. Further, support for the sun.tools.debug API was no longer supported. The new Sun product is called the "Java Platform Debugger Architecture" ("JPDA"), and it has three components:

- 5 (1) the Java Virtual Machine Debugging Interface ("JVMDI"),
which is a low-level native interface which defines what support a JVM
must provide in order to allow an application to be debugged;
- (2) the Java Debug Wire Protocol (JDWP), which allows
communication between a debugger virtual machine and a target virtual
10 machine; and
- (3) the Java Debug Interface (JDI), which is a high-level
interface, which uses JDWP and JVMDI.

New JPDA features include the ability to exclude classes while stepping,
modify Java variables, use deferred breakpoints, create watchpoints, display thread
15 names, attach to running processes, and set breakpoints at arbitrary bytecode
addresses. JPDA also allows for capturing output from applications for display by the
debugger and permits sending input from the debugger to applications. Further, JPDA
is believed to be more stable than the sun.tools.debug API.

There are some known disadvantages to using JPDA. Many functions that the
20 probe performs are more complicated to implement with JPDA, especially exception

filtering. Also, the ICAT function which allows users to load classes is not officially supported in JPDA.

- In as much as the existing ICAT does not support JDBA, and whereas JDBA provides improved functionality and stability for Java application program
- 5 development, testing, and debugging, there is a need in the art for a system and method such as ICAT which supports JDBA and its new functionality.

BRIEF DESCRIPTION OF THE DRAWINGS

The following detailed description when taken in conjunction with the figures presented herein provide a complete disclosure of the invention.

5 FIGURE 1 shows the arrangement of the components of the debugger system within a single computer system.

FIGURE 2 illustrates the arrangement of the components of the debugger system when debugging is performed from a host system while executing the target application on a remote target system.

10 FIGURE 3 shows the process of launching an application using the sun.tools.debug API.

FIGURE 4 depicts the process of launching an application by the present invention using JDPA.

15 FIGURE 5 sets forth the process of handling events using the sun.tools.debug API.

FIGURE 6 depicts the process of handling events by the present invention using JDPA.

FIGURE 7 illustrates the process of stepping through an application using the sun.tools.debug API.

20 FIGURE 8 depicts the process of stepping through an application by the present invention using JDPA.

FIGURE 9 sets forth the process of setting breakpoints in an application using the sun.tools.debug API.

FIGURE 10 shows the process of setting breakpoints in an application by the present invention using JDBA.

FIGURE 11 sets forth the process of loading classes using the sun.tools.debug API.

5 FIGURE 12 depicts the process of loading classes by the present invention using JDBA.

FIGURE 13 illustrates the process of providing a console message window for the present invention.

10 FIGURE 14 shows the use of the Event Request Manager as employed by the present invention.

FIGURE 15 is a block diagram of a representative computer in which the present invention is implemented in whole or in part, including the main components of the debugger of the present invention.

SUMMARY OF THE INVENTION

A method and system are disclosed which provide improved source-level debugging capabilities of an object-oriented application program which may include linked native language dynamic load libraries. The improved debugger is compatible with the Java Platform Debugger Architecture (JDPA), and provides new capabilities such as patching of Java variables and reading and writing strings from and to the application under test and being run by a local or remote Java Virtual Machine.

DETAILED DESCRIPTION OF THE INVENTION

The designers of JPDA intended that most debugger developers, e.g. those developers who design debug tools, to use JDI. According to the preferred embodiment of the present invention, JDI is adopted as the interface for the improved ICAT probe. By modifying the currently-available ICAT to use JPDA, users of the new ICAT have access to the new features immediately and in future ICAT releases. For clarity and distinction, the existing debugger tool will be referred to as "ICAT", and the present invention will be referred to as "ICAT2".

According to the preferred embodiment, the existing ICAT probe design as disclosed in US Patent Number 5,901,315, to Edwards, *et al.*, is used as a starting point, with modifications as disclosed herein in order to migrate the existing ICAT to support JPDA. An advantage to this approach included using existing ICAT capability for instantiating multiple debuggers on multiple platforms. However, it is possible to realize the net design disclosed herein in conjunction with the design disclosed in the Edwards patent as an entirely new program or tool.

Method names and interfaces used herein are consistent with Sun Microsystems' JDP API version 1.0, which is well known in the art and for which documentation is freely available. Sun Microsystems provides a full reference set for JDP API from their web site or for purchase in print form. Specifically, the "Java[TM] Platform Debugger Architecture Java[TM] Virtual Machine Debug Interface Reference" is hereby incorporated by reference to this disclosure.

As will be described in more detail below, one or more of the processes that comprise the "debugger" of the present invention may be carried out on a computer, or on one or more computers connected via a computer network. Referring to FIGURE 15, a computer for use in realizing the present invention is shown. The software development workstation computer 150 has a processor 151, an operating system 152, an operating system debug application programming interface ("API") (153) and a Java virtual machine (JVM) interpreter (16). The JVM has an associated Java debug API (154) to enable an application program designer to debug Java code.

Thus, for example, the computer (150) used in the present invention is any personal computer or workstation platform, such as an Intel[TM]-, PowerPC[TM]- or RISC[TM]-based computer, and that includes an operating system such as IBM[TM] OS/2[TM], Microsoft Windows[TM] '95/98/2000/NT, UNIX, or IBM AIX[TM]. Such a representative computer runs an Intel Pentium[TM] processor, OS/2 Warp Version 3 operating system, the DosDebug system API and JVM Version 1.3 or greater.

Also shown in FIGURE 15 is a high-level block diagram of the main components of the debugger of the present invention, described in more detail in the following paragraphs.

ICAT2 on a Single System

FIGURE 1 discloses the relationship (10) of the various components of ICAT2 when it is used to debug an application with the debugger running in the same

computer platform as the application. A users graphical user interface (11) is provided to allow the user to view status, data, and to input commands to load processes, set breakpoints, etc. An engine (12) is used to communicated to a daemon DLL (14), preferably by a well-known protocol such as TCP/IP messages (13). The daemon

5 DLL (14) in turn communicates to the Java Virtual Machine (16) which is running the ICAT2 probe (41). Again, communications to the JVM (16) is preferably TCP/IP messages (15). Finally, the ICAT2 probe (41) communicates to a separate instantiation of the JVM (16) running the target application (17).

In this example, both the target application (17) and the ICAT2 debugger (10)

10 are running on the same computer platform, so the communications between the various components of the system are usually internal to the computer itself.

Running ICAT2 on Two Systems

Turning to FIGURE 2, the ICAT2 user interface is run on a local computer, or “host system”, (20), while the target application (17) is run by the JVM (16) on a

15 remote “target system” (21). This is a common arrangement found during Java software development, where a number of potential target systems may be available over a computer network for a designer to use for debugging an application. The user then establishes communications with the target system using the user’s own workstation as a host system. The choice of TCP/IP for communications within the

20 single-system arrangement lends itself to dividing the components of the system between a host system (20) and a target system (21). In this case, the TCP/IP

communications (13) between the engine (12) running on the host system (20) to the daemon (14) running on the target system (21) is carried preferably over a computer network such as a Local Area Network ("LAN") or even the Internet.

Connecting to the JVM

5 First, the ICAT2 probe must establish a connection to the JVM. With the sun.tools.debug API, this was a relatively straightforward process, as shown in FIGURE 3. A Remote Debugger ("RD") object (31) was instantiated (32) by the probe (18). The parameters to the RD constructor included the name of the application to be debugged (and any parameters to be passed to the application) and a
10 pointer to the probe, which told RD that the probe would provide the required event handlers (breakpointEvent, exceptionEvent, etc.) (34).

Turning to FIGURE 4, the process of connecting to the JVM is a bit more complicated with JPDA, but JPDA supports multiple types of connectors, which facilitate attaching and listening, in addition to launching. ICAT2 supports launching
15 applications through JPDA's LaunchingConnector (LC) (44), as shown in FIGURE 4.

JPDA provides a Bootstrap object (42) that allows access to the JDI interfaces. The Bootstrap object (42) is used to obtain the VirtualMachineManager (43). The VirtualMachineManager (43) provides a list of connectors of various types. The present invention, ICAT2, iterates through the list of connectors, and searches for
20 "com.sun.jdi.CommandLineLaunch", which is stored away as a Connector object.

Launching the Application

In the older ICAT with the sun.tools.debug API, the launching process was accomplished when RD was instantiated. However, according to the present invention ICAT2 with JPDA (40), the application is launched when the probe function

5 "launch_then_attach" (46) is called. There is a set of arguments associated with each connector. The "main" argument is set to the name of the application, and the "suspend" argument is set to true. Then, ICAT2 invokes the launch method on the "LC" object. The parameters given to the method are connector arguments. The launch method returns a VirtualMachine (VM) object (45) to the probe (41).

10 The VM object (45) is similar to the RD object in the sun.tools.debug API. The VM object (45) mirrors the state of the JVM and provides the objects and methods needed for controlling the application under debug, as did the previous RD object.

Next, the probe obtains the EventRequestManager ("ERM") from the VM
15 object (45), as shown in FIGURE 14. The event handling code in the ICAT2 probe (41) makes extensive use of the ERM (80). First, an exception request (140) is created to tell the JVM to suspend the application in response to any exception that occurs. This is the default. In ICAT2, the user can later select whether to catch or ignore specific exceptions that occur within "try" blocks. Next, the modified ICAT2 creates a
20 request (141) to be notified each time a class is loaded. In JPDA, notifications of class loads are called ClassPrepareEvents. ICAT2 also request to be informed of thread death events.

For each of the request objects created above, ICAT2 must indicate that it requires the JVM to suspend the application at the time an event occurs. This is done by calling the setSuspendPolicy method on each object and passing the parameter SUSPEND_ALL. This indicates that all JVM threads are to be suspended when an event occurs. Finally, the request objects are enabled by invoking the "enable" method.

The ICAT2 probe then enters a loop in which events are received, shown in FIGURE 6 and described in more detail later. These events are processed until the class prepare event is received for the application that was launched. At this point, the ICAT2 probe (41) reads in a list of the currently loaded classes (also called "modules"). Then, a thread is spawned which enters an event retrieval loop that runs continuously as long as the probe is active. This completes the processing for the launch_then_attach function.

Obtaining Stack Information

The ICAT2 probe relies heavily on the facilities for obtaining information about the "call stack". This is handled by the ICAT2 probe similarly to the ICAT probe process. With each API, the thread object is used to obtain the data. In the sun.tools.debug API this object is the RemoteThread (RT), in JPDA it is ThreadReference (TR).

Handling Events

The JVM notifies the debugger of asynchronous occurrences by using events.

The probe MUST provide event handlers for certain events, even if it chooses not to take any action. This is true for both the sun.tools.debug API and JPDA.

5 As shown in FIGURE 5 with the sun.tools.debug API, in ICAT the only events reported were the ones for which event handlers were provided. To generate other events, the probe had to provide code to check constantly, or "poll", for new modules and threads. The ICAT probe then sent, for example, a "module loaded" event to ICAT.

10 With JPDA, there are more event handlers that must be provided by the ICAT2 probe, as disclosed in FIGURE 6. Also, a debugger designer can optionally request to be notified of a number of additional events (for example, see "Launching the Application" above). The VM object (45) has an EventQueue (46) which must be polled (60) to see these requested events, which are objects themselves. In the ICAT2
15 probe (41), events (600) are taken off the event queue (69) and checked to determine the event type. A call is then made to the proper event handler (61 through 67) for each event.

20 The processing for an event is notably different between the two APIs, sun.tools.debug and JPDA. With the sun.tools.debug API, the exception handlers were called with an RT object provided. The information about the event was obtained from the RT. However, with JPDA, the information is obtained from the event object.

Further, JPDA adds several new event handlers. The ICAT2 probe provides a function for each of these, although many of these functions may be empty.

The `stepEvent` is new with JPDA. With the `sun.tools.debug` API, the completion of a step or a breakpoint hit was considered a "`breakpointEvent`". The probe code had to sort out the situation and determine what had happened. However
5 with JPDA, there are separate handlers for `stepEvent` and `breakpointEvent`, and therefore separate event handlers (61 and 62) are provided by the ICAT2 probe (41).

With JPDA, the JVM generates an event whenever a class is loaded. This is called a `ClassPrepareEvent`. This new JPDA capability is a major improvement over
10 the `sun.tools.debug` API, through which it was necessary to poll the JVM periodically for a list of loaded classes to determine if any new classes had been loaded. Thus, a `ClassPrepare` event handler (63) is provided in the ICAT2 probe (41).

JPDA provides other new events that are not used in the preferred embodiment, including: VM start event, field watch event, VM interrupted event,
15 method entry event and method exit event. The ICAT2 probe (41), however, provides a handler for each of these events, though the handler doesn't necessarily act on the events.

Exception event handling is similar between JPDA and the `sun.tools.debug` API. As mentioned before, the information about the exception comes to the probe in
20 an `ExceptionEvent` object using JPDA, rather than from the RT object in under the `sun.tools.debug` API. An exception event handler (64) is provided in the ICAT2 probe (41) to handle receipt of an `ExceptionEvent`.

Thread death events must be handled with the sun.tools.debug API but are optional with JPDA. The sun.tools.debug API has a quit event which is analogous to the VM death event with JPDA. The architecture of ICAT2 requires notification of thread death events, so the ICAT2 probe (41) requests these events from the VM object (45). A VM death event handler (65) is provided in the ICAT2 probe (41) to handle receipt of an VMDeathEvent.

Further, thread start event and thread death event handlers (66 and 67) are provided in the ICAT2 probe (41).

Stepping

Issuing a step in the sun.tools.debug API was fairly straightforward, as shown in FIGURE 7. The thread object (70) had a "step" member function (71). The parameter to this function was a Boolean which indicated whether the step was to be a line step or an instruction step.

With JPDA, the ICAT2 probe (41) first obtains the ERM (80) from the VM object (45), as shown in FIGURE 8. From the ERM (80) a list of any outstanding step requests is obtained for the thread of interest. ICAT2 may delete any outstanding requests found. The ERM (80) has a method called "createStepRequest" (81). The parameters to this method indicate the type of step desired. The first parameter is the TR of the thread to be stepped, and the second is the size of the step. There are two sizes, "min" (minimum - which is generally an instruction step), and "line". The third parameter is depth, which may have the values "into", "over" and "out".

A count filter is added (83) to the request (82) next. The count filter is preferably set to 1, which gives the ICAT2 probe (41) a single notification when the step is completed. Finally, the request is enabled (84) and the JVM is resumed.

Obtaining Memory Usage Information

- 5 For ICAT2's memory usage window, few changes from ICAT were required. The information about free memory and total memory was available to ICAT from the RD with the sun.tools.debug API, which is now available to ICAT2 from an object called "Runtime" with JPDA. As such, ICAT2 obtains memory usage information from the JDPA Runtime object.

10 Obtaining Thread Names

JPDA allows ICAT2 to obtain the name of any thread, whereas the sun.tools.debug API RD did not support this. The TR object has a method that returns the name, which is employed by ICAT2 to obtain the name of any thread.

Filtering Exceptions

- 15 ICAT2 allows the user to choose whether or not to have the debugger report exceptions that occur within "try" blocks. When an exception occurs outside a "try" block, execution always stops and the exception is reported to the user.

With the sun.tools.debug API, catching or ignoring exceptions in ICAT was a relatively simple process. The RemoteClass (RC) object had both an

“ignoreExceptions” and a “catchExceptions” method. In order to obtain an exception object, the “findClass” method on RD was called. This loaded the class if it is not already loaded.

The exception filtering process for ICAT2 with JDPA is more complicated than with sun.tools.debug. If the exception class of interest is not loaded, it must be loaded according to a process described below. If a request object for the exception already exists, ICAT2 disables and deletes it so that a new one can be created. When ICAT2 calls “createExceptionRequest”, one of the parameters to the method is a Boolean which indicates whether caught exceptions (exceptions within a “try” block) are to be reported or ignored. The newly created “ExceptionRequest” object is enabled after setting the suspend policy.

Loading Classes

There are several circumstances in which a debugger such as ICAT or ICAT2 must force the loading of a class. ICAT provided a function with which the user could request the forced loading of a class. As shown in FIGURE 11, this was relatively uncomplicated with the sun.tools.debug API since the “findClass” method (92) automatically loads a class if it is not already loaded.

However, class loading is a multi-step process with JPDA, as shown in FIGURE 12. ICAT2 first must obtain a reference to java.lang.ClassLoader (120). ICAT2 then has to obtain a reference to the method “getSystemClassLoader” (121), which is then invoked. In order to do so, ICAT2 must have a reference to an

application thread. Unfortunately, this does not work for any thread other than the main thread, so a reference to the main thread (122) is obtained by the ICAT2 probe. Since the main application thread can terminate before the application terminates, it is possible that the user will see a situation in which it is impossible to force the loading
5 of a class.

Obtaining Thread Status

JPDA has the ability to obtain the status of any application thread, such as "running", "sleeping", "waiting", "not started", etc. The TR object has a member function called "status" which returns this information when invoked by the ICAT2
10 probe.

Providing and Displaying a Console Message Window

The move to JPDA allows ICAT2 to provide a console window, as shown in FIGURE 13. This permits users to see output printed by the application, and to send data to the application. In order to allow writing data to the application, the ICAT2
15 probe (41) first obtains the "Process" object (130) from the VM object (45). The "OutputStream" (131) is obtained from the "Process" object (130). The OutputStream is used to instantiate a "BufferedWriter" (133). The "write" method (136) on the "BufferedWriter" object (133) is called in order to write data to the application from the console message window.

For obtaining data that is printed by the application and presenting it on the console message window, a "BufferedReader" object (134) is created in a manner similar to that used to create a BufferedWriter, including obtaining the Process object (130) from the VM object (45), obtaining the InputStream object (132) from the Process object (130), instantiating a BufferedReader object (134), and using the "read" method (135) to retrieve data from the application.

Setting Breakpoints

Turning to FIGURE 9, the process employed by the ICAT probe (18) with the sun.tools.debug API to set a breakpoint is fairly simple, which involved calling the findClassmethod (92) of the RD (31) and the "setBreakpointLine" method (91) on the class where the breakpoint was to be placed. The line number provided was the source line. There was no provision for setting breakpoints at an arbitrary bytecode address.

The ICAT2 probe (41) with JPDA first obtains a list of the loaded classes (102), as shown in FIGURE 10. If the class where the breakpoint is to be placed is not loaded, the setting of the breakpoint is deferred by adding it to deferred breakpoint table (101) to be installed later. If the class is loaded, the line number information is obtained from the class object (103). Then the breakpoint request (106) is instantiated (107) by the ERM (80), its suspend policy is set (108), and the request is enabled (109).

Clearing Breakpoints

Clearing a breakpoint was similarly straight forward with the sun.tools.debug API for ICAT. The method that was used was "clearBreakpointLine".

- For ICAT2 with JDPA, a list of the active BreakpointRequests must first be
- 5 obtained. If ICAT2 finds one that matches the user's requested breakpoint deletion, that BreakpointRequest object is deleted using a JDPA API command.

Halting an Application

- In order to halt the application, ICAT2 obtains a list of the current application threads by calling the VM's "allThreads" method. ICAT2 then iterates through this list
- 10 and invokes the JDPA API "suspend" method on each thread that is not already suspended.

Requesting Thread Information

- When ICAT2 obtains the list of threads from the VM object (45), it checks to make sure that the thread group is the main group, which is the group of application
- 15 threads. Any threads that are in the main group are added to the list that is returned.

Resuming Execution

The "resume" command to the probe is a request to restart execution of the application. Resuming is one area that may be simpler with JPDA than with the sun.tools.debug API.

ICAT invoked one of three methods through the sun.tools.debug API: the "run" or "cont" method on the RD object, or the "resume" method on any particular RemoteThread object.

ICAT2 using the JPDA API reviews the current list of step requests and
5 deletes any that are currently active. Then, ICAT2 invokes the "resume" method on the JVM object.

Excluding Classes

During migration from ICAT with sun.tools.debug to ICAT2 with JPDA, a new feature was brought to ICAT2: the ability to exclude classes when stepping. This
10 allows users to prevent ICAT2 from stopping in certain classes when stepping. It is possible to exclude entire sets of classes by use of an asterisk. For example, to exclude all classes beginning with "abc.def", the user sets the environment variable
CAT_CLASS_EXCLUDE=abc.def.*. This would exclude abc.def.ghi, abc.def.jkl, etc. The ICAT2 probe excludes classes by invoking the "addClassExclusionFilter" method
15 on the step request object.

Evaluating Expressions

For expression evaluation, no major changes were needed to the probe to migrate from the sun.tools.debug API to JPDA. A line-by-line replacement of RD objects and methods with VM objects and methods is sufficient. However, the ICAT2
20 user is now able to modify Java variables.

Initializing Objects

The “ExpressionEvaluator” and “ExpressionServices” constructors are initialized with a VM object in place of an RD object.

Accessing and Querying of Debuggee Classes

- 5 The VM contains methods (classesByName and allClasses) for returning ReferenceType objects in the debuggee. The ReferenceType object contains the necessary methods for accessing information about the class under test such as class methods, fields, types, values, static, final, etc.

Adding New Function

- 10 Modifying Symbols. JPDA provides “setValue” and “mirrorOf” utilities. SetValue performs modification of static and instance fields, strings, array elements, and Booleans. MirrorOf returns an access reference to the type being modified. A new method, update_value, was added to the probe which utilizes both of these utilities to change dynamically and return new field values to the user during a debug
15 session.

Evaluating Strings. Although ICAT provided string evaluation, ICAT2 with JPDA includes a StringReference interface, which extends ObjectReference. This means that ICAT2 can treat the string as a class, and therefore provide the user with much more information pertaining to the string. The RmtStringValue class was

subsequently eliminated, with the new string evaluation code included in
RmtObjectValue class.

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226